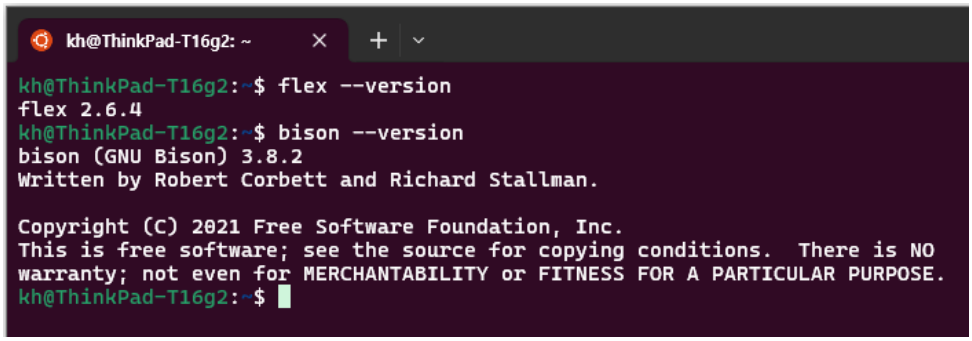
컴파일러 과제7
보고서
(1) - 제작

20181755 이건희

2024년 12월

1. 개발 환경

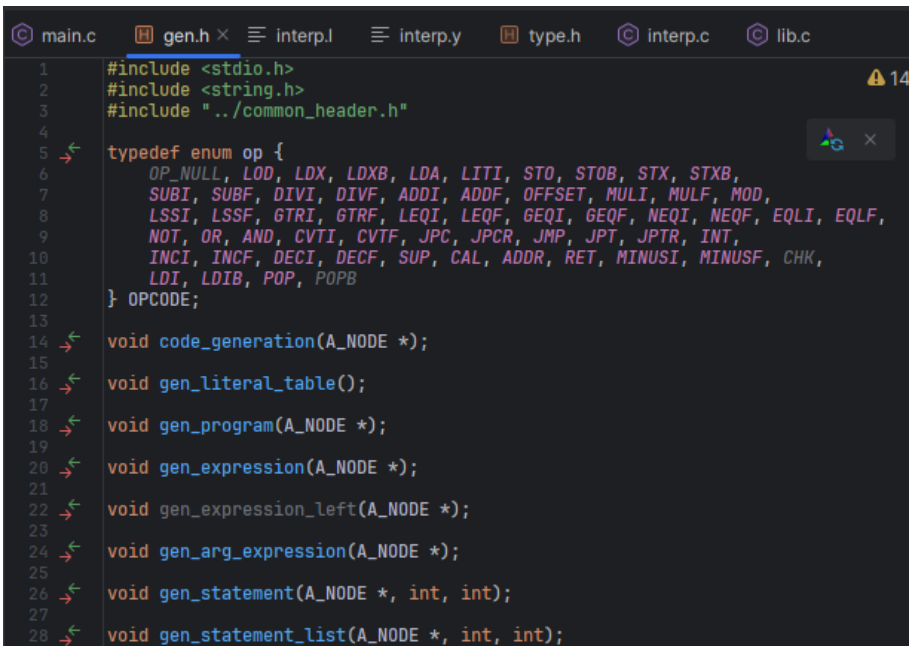
기존과 동일한 wsl2 상의 Ubuntu 22 LTS, gcc 11, gdb 12이다.



lex로 flex, yacc로 bison을 사용하였다.

2. gen.c, gen.h

적을 양이 많아 수정 사항에 포함하는 것이 아닌 새로운 문단으로 분리한다.



우선 지금까지 해온 것처럼 .h와 .c로 분리하였다.

함수를 완성하기 전에 파일에 프린트하는 함수들을 간단히 짚고 가겠다. 강의에서 상정한 가상기계에서는 inst의 형식을 <opcode, a, b>로 정했다. 가상 기계의 inst들이 어떻게 작동하는지는 이미 많은 공부를 했기에 설명을 생략한다.

gen_code_xxx() : “opcode, a, b”를 파일에 적는다. _i, _f, _s, _l은 세 번째 파라미터(<b>)에 들어가는 타입에 따라 결정된다. l는 int, f는 float, s는 cal종류, l은 레이블이다.
gen_label_xxx() : 강의교안이나 책에서는 gen_label로 나와있었으나, 실제 코드를 살펴보니 gen_label_number 혹은 gen_label_name으로 사용해야 했다. 이들은 각각 숫자, 이름으로 새로운 레이블을 생성한다. 후자는 어셈블리어의 일관성을 위해 오직 함수에서만 사용하는 것으로 약속하겠다.

2-1. 선언문

```
378  void gen_declaration(A_ID *id) {
379      int i;
380      A_NODE *node;
381      switch (id→kind) {
382          /** 선언문은 본 과제에서 하지 않기로 약속했다 */
383          case ID_VAR:
384              /** 초기화는 강의에서 아예 다루지 않는다 약속했다 */
385              if(id→init){
386                  if(id→level == 0){
387                      // gen_initializer_global();
388                  }
389                  else{
390                      // gen_initializer_local();
391                  }
392              }
393              break;
394          case ID_FUNC:
395              /** prototype은 하지 않고, body(type table→expr)가 존재할 경우만 한다 */
396              if (id→type→expr) {
397                  gen_label_name(id→name);
398                  gen_code_i(INT, 0, id→type→local_var_size);
399                  gen_statement(id→type→expr, 0, 0);
400                  gen_code_i(RET, 0, 0);
401              }
402              break;
```

```
477  /** 초기화는 강의에서 아예 다루지 않는다 약속했다 */
478  void gen_initializer_global(A_NODE *node, A_TYPE *t, int addr) {
479  }
480
481  void gen_initializer_local(A_NODE *node, A_TYPE *t, int addr) {
482  }
```

변수의 초기화는 과제4(첫 c언어 파서)부터 아예 다루지 않기로 약속했다. 또한 10장 강의에서 선언문은 함수는 프로토타입이 아닌 타입 테이블에 바디가 존재할 경우만 코드를 생성하고, 이 외에는 전부 다루지 않기로 약속했다.

2-2. 수식

어려워서 부록을 참고하였고, 교재의 9장을 보며 사후적으로 이해하는 방향으로 과제를 진행했다.

```
        break;

        /** 구조체는 이번 과제에서 하지 않는다 약속했다 */
    case T_STRUCT:
    case T_UNION:
        break;

    case ID_ENUM_LITERAL:
        gen_code_i(LIT, 0, id→init);
```

```
        /** 구조체의 참조수식은 하지 않기로 약속했다. */
    case N_EXP_STRUCT :
        // not implemented yet
    case N_EXP_ARROW:
        // not implemented yet
        break;
```

수업중에 구조체, 구조체의 참조연산을 하지 않기로 약속했기에 해당 case는 비워두었다.

```
case N_EXP_IDENT :
    id = node->clink;
    t = id->type;

    switch (id->kind) {
        case ID_VAR:
        case ID_PARM:
            switch (t->kind) {
                case T_ENUM:
                case T_POINTER:
                    gen_code_i(LOD, id->level, id->address);
                    break;
                case T_ARRAY:
                    if (id->kind == ID_VAR) {
                        gen_code_i(LDA, id->level, id->address);
                    }
                    else {
                        gen_code_i(LOD, id->level, id->address);
                    }
                    break;

                    /** 구조체는 이번 과제에서 하지 않는다 약속했다 */
                case T_STRUCT:
                case T_UNION:
                    break;

                case ID_ENUM_LITERAL:
                    gen_code_i(LITI, 0, id->init);
                    break;
            }
    }
    break;
```

<명칭>
변수와 파라미터의 경우에만 코드를 생성한다.
enum이나 pointer의 경우 LOD
array가 변수일 경우 LDA, parm일 경우 LOD
literal일 경우 LITI
inst를 실행한다.

```
 99             case N_EXP_INT_CONST :
100                 gen_code_i(LITI, 0, node->clink);
101                 break;
102             case N_EXP_FLOAT_CONST :
103                 i = node->clink;
104                 gen_code_f(LITI, 0, literal_table[i].addr);
105                 break;
106             case N_EXP_CHAR_CONST :
107                 gen_code_i(LITI, 0, node->clink);
108                 break;
```

<상수, 리터럴>
정수, 문자 상수의 경우 정수값을 LITI
실수 상수의 경우 상수셩역의 실수값의 주소에서 LOD

```
109             case N_EXP_STRING_LITERAL :
110                 i = node->clink;
111                 gen_code_i(LDA, 0, literal_table[i].addr);
```

스트링 리터럴일 경우 상수영역의 스트링 주소를 LDA
inst를 실행한다.

```
112             case N_EXP_ARRAY :
113                 gen_expression(node->llink);
114                 gen_expression(node->rlink);
115                 t = node->type;
116                 if (t->size > 1) {
117                     gen_code_i(LITI, 0, t->size);
118                     gen_code_i(MULI, 0, 0);
119                 }
120                 gen_code_i(OFFSET, 0, 0);
121                 if (!isArrayType(t)) {
122                     i = t->size;
123                     if (i == 1) {
124                         gen_code_i(LDIB, 0, 0);
125                     }
126                     else {
127                         gen_code_i(LDI, 0, (i % 4) ? (i / 4 + 1) : (i / 4));
128                     }
129                 }
130                 break;
```

<배열 참조>

llink 수식의 값(주소), rlink 수식의 값(offset)을 계산하여

element의 size가 1보다 클 경우 LITI <element size> 하여 MULI 한 것을 OFFSET 한다.

```
131             case N_EXP_FUNCTION_CALL :
132                 t = node->llink->type;
133                 i = t->element_type->element_type->size;
134                 if (i % 4) {
135                     i = i / 4 * 4 + 4;
136                 }
137                 if (node->rlink) {
138                     gen_code_i(INT, 0, 12 + i);
139                     gen_arg_expression(node->rlink);
140                     gen_code_i(POP, 0, node->rlink->value / 4 + 3);
141                 }
142                 else {
143                     gen_code_i(INT, 0, i);
144                 }
145                 gen_expression(node->llink);
146                 gen_code_i(CAL, 0, 0);
147                 break;
```

<함수 호출>

함수의 activation record의 초기 값을 가져와서

rlink(파라메터)가 있을 경우 이를 더하여 INT 후 다음 activation record의 주소값에 입력 후 POP

없을 경우 INT

한뒤 CAL을 한다.

```
280             case N_EXP_AMP :
281                 gen_expression_left(node->clink);
282                 break;
```

<unary 수식 중 &>

lvalue expression이어야 하기에 gen_expression_left를 호출한다. 바로 다음 장에서 설명하겠다.

이외에는 읽어보면 상식적으로 이해가 가능한 내용들이라 생략한다.

2-3. 좌변 수식

```
452 void gen_expression_left(A_NODE *node) {
453     A_ID *id;
454     A_TYPE *t;
455     switch (node->name) {
456         case N_EXP_IDENT :
457             id = node->clink;
458             t = id->type;
459             switch (id->kind) {
460                 case ID_VAR:
461                 case ID_PARM:
462                 case ID_FUNC:
463                     break;
464             }
465             break;
466         case N_EXP_ARRAY :
467             gen_expression(node->llink);
468             gen_expression(node->rlink);
469             if (node->type->size > 1) {
470                 gen_code_i(LITI, 0, node->type->size);
471                 gen_code_i(MULI, 0, 0);
472             }
473             gen_code_i(OFFSET, 0, 0);
474             break;
```

lvalue expression은 수정 가능한 값이어야 한다. 기말고사에도 나온 내용이지만, lvalue가 될 수 있는 것은 변수나 파라미터의 명칭, 구조체의 멤버를 필드나 포인터로 참조 (단, 배열형은 안된다), 배열 참조가 있다. gen_expression_left는 수식의 주소에 해당하는 패턴을 생성한다.

```
476             /** 구조체의 참조수식은 하지 않기로 약속했다. */
477         case N_EXP_STRUCT :
478             // not implemented yet
479             break;
480         case N_EXP_ARROW :
481             // not implemented yet
482             break;
483 
484         case N_EXP_STAR :
485             gen_expression(node->clink);
486             break;
```

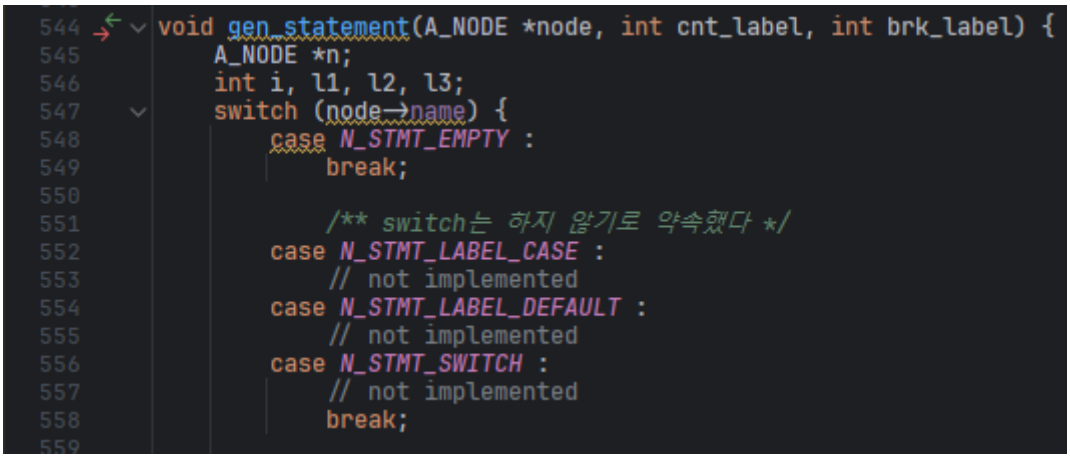
이중 구조체 멤버를 참조하는 것은 하지 않기로 약속했기에, 제외한다.

```
488             /** lvalue가 될 수 없는 수식!! */
489         case N_EXP_INT_CONST :
490         case N_EXP_FLOAT_CONST :
491         case N_EXP_CHAR_CONST :
492         case N_EXP_STRING_LITERAL :
493         case N_EXP_FUNCTION_CALL :
494         case N_EXP_POST_INC :
495         case N_EXP_POST_DEC :
496         case N_EXP_PRE_INC :
497         case N_EXP_PRE_DEC :
498         case N_EXP_NOT :
499         case N_EXP_MINUS :
500         case N_EXP_SIZE_EXP :
501         case N_EXP_SIZE_TYPE :
502         case N_EXP_CAST :
503         case N_EXP_MUL :
504         case N_EXP_DIV :
505         case N_EXP_MOD :
506         case N_EXP_ADD :
507         case N_EXP_SUB :
508         case N_EXP_LSS :
509         case N_EXP_GTR :
510         case N_EXP_LEQ :
511         case N_EXP_GEQ :
512         case N_EXP_NEQ :
513         case N_EXP_EQL :
514         case N_EXP_AMP :
515         case N_EXP_AND :
516         case N_EXP_OR :
517         case N_EXP_ASSIGN :
518             gen_error(12, node->line, NULL);
519             break;
```

lvalue가 되지 못하는 수식들에 대해서는 경고를 띄워야 한다.

2-4. 명령문

강의자료를 참고하여 완성하였다.



```
544 void gen_statement(A_NODE *node, int cnt_label, int brk_label) {
545     A_NODE *n;
546     int i, l1, l2, l3;
547     switch (node->name) {
548         case N_STMT_EMPTY :
549             break;
550 
551             /** switch는 하지 않기로 약속했다 */
552         case N_STMT_LABEL_CASE :
553             // not implemented
554         case N_STMT_LABEL_DEFAULT :
555             // not implemented
556         case N_STMT_SWITCH :
557             // not implemented
558             break;
559 
```

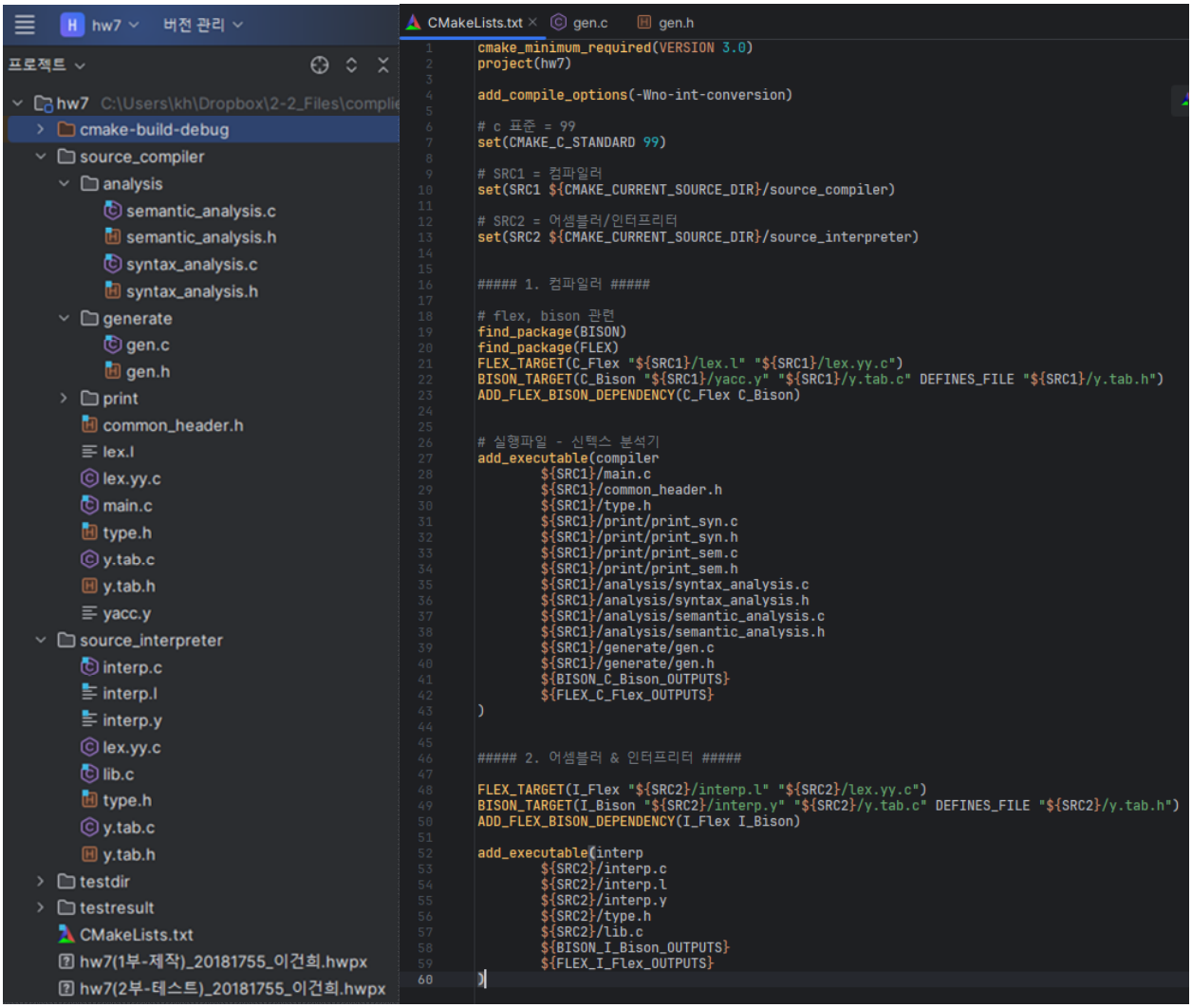
수업 중에 switch는 하지 않기로 약속했기에 해당 case는 비워두었고, 테스트에서도 이는 제외하도록 한다.

이외에는 읽어보면 상식적으로 이해가 가능한 내용들이라 생략한다.

3. 그 외 파일들

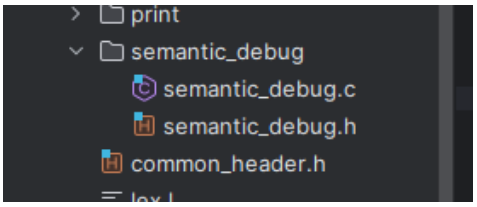
과제 6의 코드를 거의 그대로 사용하되 약간의 수정이 있었다.

3-1. CMakeLists.txt



새롭게 추가된 파일과 디렉터리 구조 변경에 따라 적절히 구성하였다.

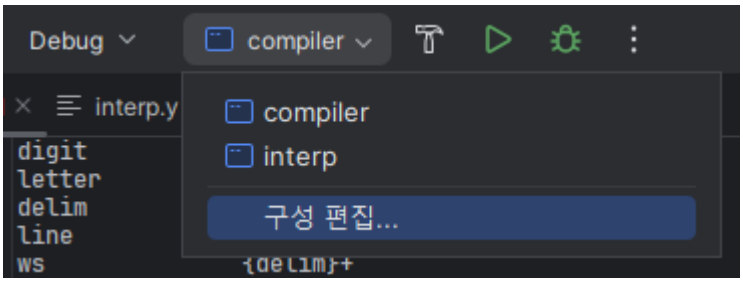
3-2. semantic analysis의 debug 관련 삭제

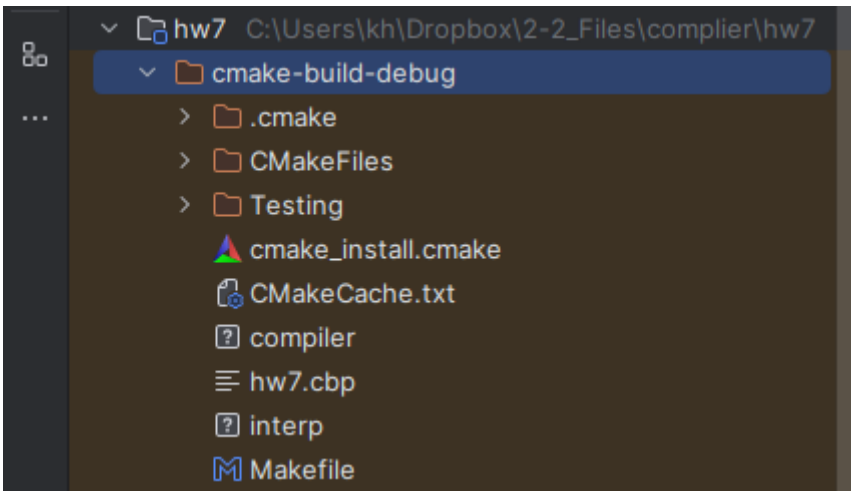



```
1357        return result;
1358    }
1359
1360    void semantic_error(int i, int l, char *s) {
1361    #if DEBUG
1362        earray[i] = 1;
1363        printf("error case %d - ", i);
1364    #endif
1365        semantic_err++;
1366        printf("*** semantic error at line %d: ", l);
```

이전 과제의 semantic analysis에서 디버그를 위해 추가한 코드들을 전부 삭제했다.

3. 빌드





이전 과제처럼 ide에서 CMake를 사용하여 빌드하였다. 각각의 소스파일들로 컴파일러, 인터프리터 두 개의 실행파일이 나온다. 큰 오류 없이 빌드가 되었다.

4. 테스트

양이 많아서 편의를 위해 다른 파일에 적겠습니다.

5. 소스코드

부록 없이 따로 파일을 첨부하겠습니다.